Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Zupan

Visokorazpoložljvi sistem za nadzor in opazovanje

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: dr. Andrej Brodnik

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Sodobni IKT sistemi postajajo vedno bolj zapleteni in posledično zahtevajo učinkovite sisteme za nadzor in opazovanje. Različni proizvajalci ponujajo relativno drage in zahtevne rešitve v ta namen. Ker pa je nadzor in opazovanje samo ena od storitev, ki jo uporabnik pričakuje od sistema, je posledično razumljivo, da pričakuje njeno robustnost.

V diplomski raziskavi preglejte ustrezne sisteme za nadzor in opazovanje ter preučite njihovo arhitekturo. Poleg tega preučite sisteme, ki omogočajo učinkovit preklop storitev ter tako naredijo preklapljano storitev robustno oziroma visokorazpoložljivo. Kot rezultat dela namestite in ustrezno nastavite konfiguracijo visokorazpoložljivega sistema za nadzor in opazovanje ter ovrednotite njegovo delovanje.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nejc Zupan, z vpisno številko **63100312**, sem avtor diplomskega dela z naslovom:

Visokorazpoložljvi sistem za nadzor in opazovanje

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Andreja Brodnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 1. avgust 2014 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

kratica	okrajšuje
SNMP	Simple Network Management Protocol
RMON	Remote Network Monitoring
SLA	Service Level Agreement
MIB	Management Information Base
SMI	Structure of Management Information
CMIP	Common Management Information Protocol
LAN	Local Area Network
WAL	Wide Area Network
TMN	Telecommunications Management Network
API	Aplication Programming Interface
B2B	Business to Business
WAL	Write Aheah Log
2PC	Two-phase Commit
APT	Advanced Packaging Tool
LSB	Linux Standard Base
OCF	Open Cluster Framework

Povzetek

S porastom uporabe spletnih tehnologij in storitev je nadzor omrežij postal nuja. Vse več storitev je danes na voljo preko spleta, s čimer se je povečalo tudi število uporabnikov. Ker želimo storitvam zagotoviti čim boljšo stabilnost, se poslužujemo visokorazpoložljivih in nadzornih sistemov.

V diplomskem delu smo obravnavali nekaj nadzornih protokolov in sistemov za nadzor omrežij, na podlagi katerih smo izbrali najprimernejše in jih podrobneje opisali. Temu opisu sledi eksperimentalni del, v katerem smo postavili prototip visokorazpoložljivega nadzornega sistema, ki tudi v primeru preklopa uporabniku nudi nemoteno delovanje. Naša rešitev vsebuje nadzorni sistem Zabbix, ki z agenti komunicira preko izbranega SNMP protokola. Za nastavitev preklopa smo uporabili odprtokodne programe Pacemaker, Corosync in Heartbeat, za serviranje spletnega vmesnika pa Nginx. Podatki se hranijo v podatkovni bazi PostgreSQL, ki deluje v aktivno-pasivnem načinu in prav tako podpira preklop med vozlišči.

Ključne besede: visokorazpoložljivi sistemi, Zabbix, preklop, SNMP, nadzorni sistem.

Abstract

As we see an increase in the use of web services, having a control over networks became a priority. More and more services are available online and consequently, the number of users has increased. Since our aim is to get more stability, we are using high available monitoring system.

In this thesis we addressed few protocols and systems for monitoring networks, based on which we chose the most appropriate ones and described them in a greater detail. Following that description is experimental part in which we set up a high available monitoring system that ensures smooth user interaction even during failover to the other node. The solution includes monitoring system Zabbix which communicates with the agents via the selected SNMP protocol. For failover we used open-source programs like Pacemaker, Corosync and Heartbeat, while we used Nginx for monitoring web page. Data are stored in the PostgreSQL database that operates in active-passive mode and also supports failover between nodes.

Keywords: high availability, Zabbix, failover, SNMP, monitoring system.

Poglavje 1

Uvod

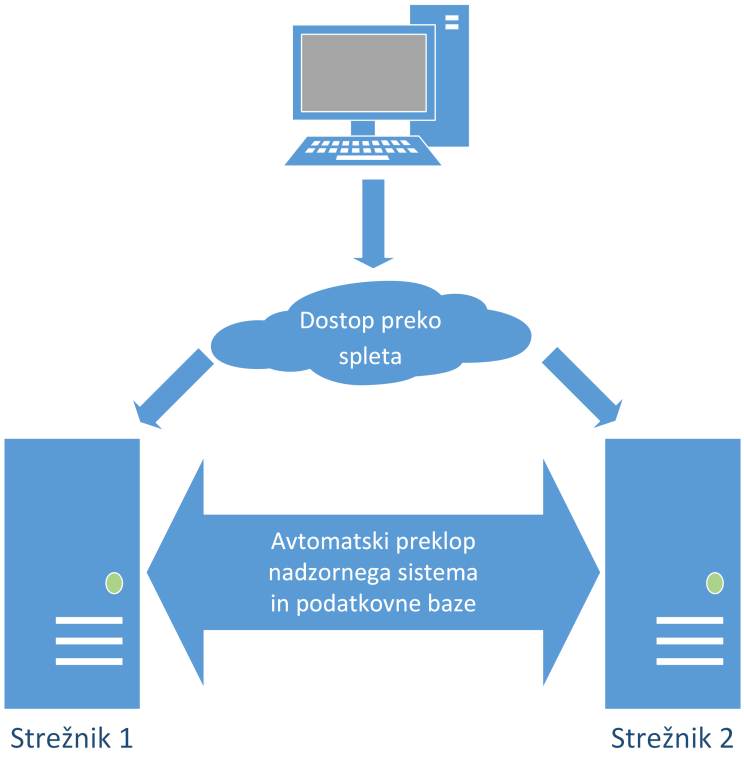
Danes skoraj vsi uporabljamo informacijske naprave. Naj bo to za razvedrilo, komunikacijo z bližnjimi ali v poslovne namene. V nobenem primeru se ne moremo izogniti spletu in povezavi v omrežja. Omrežja so danes že tako kompleksna in celovita, da brez nadzorovanja in upravljanja omrežja ne bi bila uporabna.

Upravljanje omrežij je obstajalo še preden so bili protokoli za upravljanje in nadzor standardizirani, vendar je bilo to delo omrežnih administratorjev in tako se je upravljanje razlikovalo od organizacije do organizacije [15]. V malih podjetjih se še danes pogosto srečujemo z rešitvami, ki so narejene po meri dotičnh omrežij. Za premostitev tega problema obstajajo številne komercialne rešitve velikih korporacij, kot na primer Cisco Prime, IBM Tivoli Monitoring, HP Network Node Manager i, ki nudijo tako celovito paleto orodij, analiz in vizualizacij, kot tudi visokorazpoložljivost, vendar cenovno niso dosegljive vsakomur.

1.1 Cilji

Namen diplomske naloge je z uporabo protokolov za nadzor omrežij in odprtokodnim nadzornim sistemom postaviti visokorazpoložljivo storitev, katere arhitektura je vidna na sliki 1.1, ki bo konkurenčna zgoraj naštetim produk-

tom in bo med preklopom uporabnikom nudila transparentnost.



Slika 1.1: Visokorazpoložljiv sistem.

1.2 Struktura naloge

V poglavju 2 bomo opisali nekaj protokolov, ki služijo nadzoru omrežij, ter izbrali najprimernejšega. V naslednjem poglavju 3 bomo obravnavali odprtokodne nadzorne sisteme, ki podpirajo izbran omrežni protokol in izbrali najprimernejšega. Izbran sistem bo osnova visokorazpoložljivemu sistemu, ki ga bomo v poglavju 5 realizirali z odprtokodnimi komponentami, opisanimi v poglavju 4.

Diplomska naloga je delno izdelana kot del projekta Iskratel RV-IT-FRI 2013-1.

Poglavje 2

Protokoli za upravljanje omrežij

Namen upravljanja in nadzora omrežij je zagotoviti kvaliteto storitve, kakršno pričakuje uporabnik. V ta namen je z uporabnikom pogosto dogovorjen SLA (*Service Level Agreement*) [13], ki določa delovanje storitve in njeno dostopnost.

V tem poglavju bom opisal tri protokole, ki se jih uporablja za upravljanje in nadzor omrežij, SNMP, CMIP in RMON.

2.1 SNMP

SNMP (*Simple Network Management Protocol*) je protokol za upravljanje, nadzor in izmenjavo omrežnih informacij med omrežnimi napravami. Ta pogosto predstavlja osnovo za mrežno arhitekturo [20]. Tekom let je zaradi enostavnosti in hitre implementacije agentov postal de facto standard na tem področju.

2.1.1 Arhitektura SNMP protokola

Sestavljena je iz treh glavnih elementov [16]:

- `upravitelj` ima nalogo komunikacije s povezanimi napravami, ki izvajajo SNMP agente. Podatke pridobiva s periodičnim izpraševanjem agentov. Slabost tega načina je zakasnitev med trenutkom, ko se dogodek zgodi in trenutkom, ko je zabeležen.

- `agenti` so sistemi za zajem informacij na napravah oziroma iz storitev. Ob sprejemu zahtevka upravitelja se ti odzovejo z zajetimi informacijami.

- `MIB` (*Management Information Base*) je podatkovna struktura, ki opisuje ključne podatke o entitetah, ki jih agenti posredujejo upraviteljem.

Vsi podatki o stanju naprav, ki jih agenti posredujejo upravljalcem in zapisi o izmenjanih vrednostih, se zbirajo v bazi MDB na strani upravljalca.

Za izmenjavo podatkov med upraviteljem in agenti obstajajo tri inačice SNMP sporočilnih protokolov [20].

2.1.2 SNMPv1

SNMPv1 [3] je prvotni sporočilni SNMP protokol in je izredno preprost. Varnost sporočila temelji na imenu skupine, ki služi kot geslo. Če upravitelj pošlje zahtevek s pravilnim imenom skupine, se bo naslovljeni agent odzval, drugače ne. SNMPv1 ne podpira nobene možnosti enkripcije sporočil.
Definirani so naslednji tipi sporočil, s primerom na sliki 2.1.

`Smer upravitelj - agent`:

- *get* zahtevek zahteva vrednosti za en ali več MIB objektov

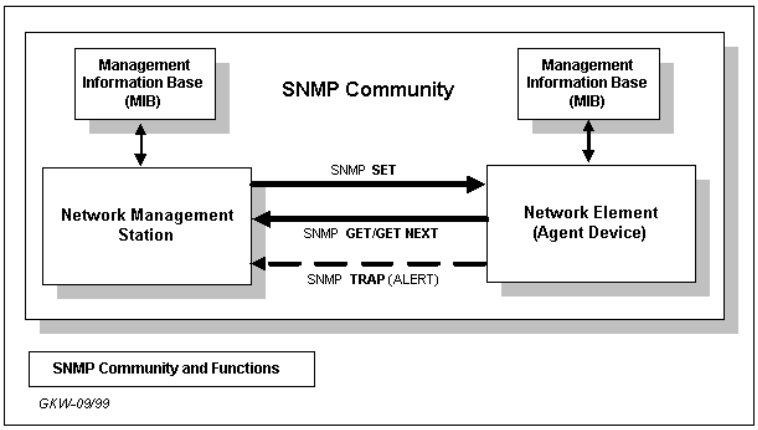
- *getnext* zahtevek je namenjen za uporabo po uporabi prvotnega *get* zahtevka, kar omogoča vrstni red

- *set* zahtevek omogoča spreminjanje vrednosti v enem ali več MIB objektov

`Smer agent - upravitelj`:

- *getresponse* sporočilo vsebuje podatke, ki so bili zahtevani v *get* ali *getnext* zahtevku

- *trap* sporočila ne potrebujejo zahtevka s strani upravitelja, za razliko od *notifications* in *events*. Agent jih posreduje upravitelju v primeru pomembnih dogodkov

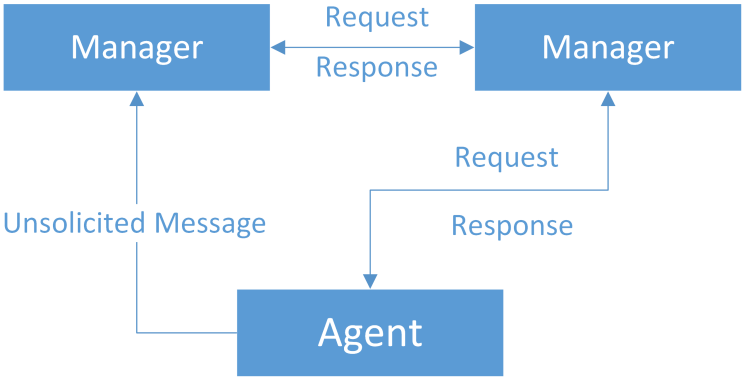


Slika 2.1: Promet SNMPv1 protokola.[1]

2.1.3 SNMPv2

Protokol SNMPv2 je nadgradnja in nadomestilo protokola SNMPv1. Sprva je bil glavni namen nadgradnja varnosti, ki pa je ta verzija ni doprinesla. Dodana je bila komunikacija med upravitelji (vidno na sliki 2.2) in izboljšana učinkovitost. Kljub vsemu verzija ni bila sprejeta. Pojavilo se je veliko eksperimentalnih verzij, ki so bile poskus sprejetja protokola SNMPv2. Danes SNMPv2 enačimo s protokolom SNMPv2c (*Community-based SNMPv2*) [7], saj je najbolj podprt v komercialnih produktih. Še vedno vsebuje enako stopnjo varnosti kot protokol SNMPv1, ima pa dodane nove tipe sporočil, ki so optimizacija obstoječih.

[1] http://www.wtcs.org/snmp4tpc/images/snmp.gif



Slika 2.2: Komunikacija med upravitelji v SNMPv2[2].

Dodani tipi sporočil:

Smer upravitelj - agent:

- *getbulk* sporočilo je optimizacija učinkovitosti *get* in *getnext* zahtevkov, saj zmanjša količino ponavljajočih se zahtevkov. Namenjen je pridobivanju velikih količin podatkov (tabel)
- *inform* zahtevek je nadgradnja tipa sporočila *trap*. Razlika je v potrditvi, ki jo upravitelj pošlje agentu ob prejemu sporočila

Smer agent - upravitelj:

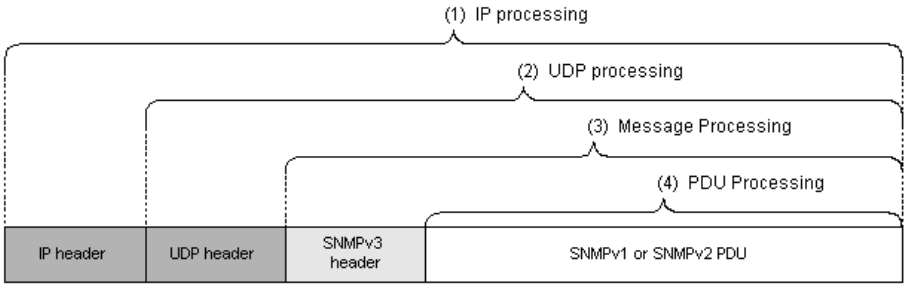
- *getbulkresponse* je odgovor na *getbulk* zahtevek
- *v2trap* je drugače enkodirana oblika sporočila *trap* v protokolu *SNMPv1*, saj je slednji uporabljal drugačen format kot ostala sporočila. Z verzijo SNMPv2 je format poenoten s formatom GET sporočila

2.1.4 SNMPv3

SNMPv3 [17] je zadnja verzija SNMP standarda. S to verzjo je SNMP protokol dodatno varnost, ki omogoča avtentikacijo in varen prenos.

[2]http://www.cse.wustl.edu/ jain/cis788-97/ftp/net_monitoring/fig1.gif

Iz slike 2.3 je razvidno, da struktura sporočil ostaja enaka, dodana je le avtentikacija in enkripcija sporočil.



Slika 2.3: Datagram sporočila verzije SNMPv3[3].

SNMPv3 doda 3 stopnje varnosti:

- *NoAuthNoPriv* omogoča prenos brez avtentikacije in enkripcije.
- *AuthNoPriv* zahteva avtentikacijo, vendar vsebina ni kriptirana.
- *AuthPriv* zahteva avtentikacijo in zagotavlja kriptiran prenos.

SNMPv3 omogoča dodeljevanje pravic na nivoju uporabnika in skupine.

2.1.5 MIB (Management Information Base)

MIB datoteke so osnova SNMP protokola. Vsebujejo hierarhično (drevesno) zgradbo, kjer je vsak vnos naslovljen z enolično oznako OID in imenom. Podatki, ki se prenašajo med agenti in upraviteljem (*trap* in *data*), so definirani s SMI (*Structure of Management Information*) ogrodjem [20]. Zapisi se lahko razlikujejo glede na verzijo SMI ogrodja, s katerim so definirani. Danes poznamo dve verziji; SMIv1, ki je izšla skupaj z SNMPv1 in SMIv2, ki je

[3]http://www.tml.tkk.fi/Opinnot/Tik-110.501/1999/papers/management/pdu_encapsulation.gif

izšla z verzijo SNMPv2 protokola. Poleg tega je običajno, da imajo velike korporacije definirane svoje SMI standarde, na primer *CISCO-SMI* [20].

SMIv2 MIB ne vsebuje veliko generalnih sprememb glede na SMIv1, ampak le boljšo dokumentacijo, več podatkovnih tipov in bolj konsistentno sintakso. Več podrobnosti lahko bralec najde v [20]. Možna je tudi obojesmerna pretvorba med SMIv1 in SMIv2 objekti, z nekaj omejitvami pri pretvorbi iz SMIv2 v SMIv1, saj SMIv1 ne podpira podatkovnega tipa *Counter64*, ki je bil predstavljen šele v SMIv2 [8].

2.2 CMIP

CMIP (*Common Management Information Protocol*) je *OSI management* protokol standard in je namenjen upravljanju LAN in WAN omrežij.

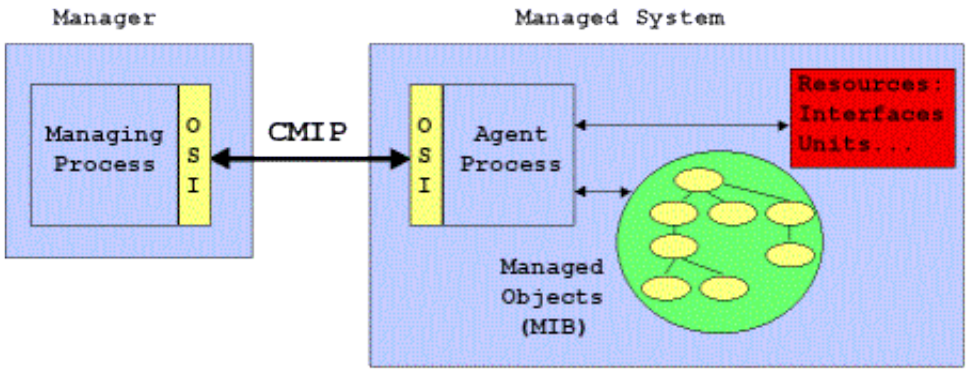
Njegova implementacija je objektno orientirana in pokriva vseh sedem nivojev *ISO/OSI* modela. S tem omogoča popoln nadzor vseh nivojev prenosa, kar naredi implementacijo rešitev izredno kompleksno in omeji možnost uporabe le na naprave, ki ustrezajo vsem zahtevam za implementacijo OSI modela. CMIP protokol srečamo predvsem pri nadzoru vozlišč v telekomunikacijskih omrežijh TMN (*Telecommunications Management Network*) [15].

2.2.1 Arhitektura CMIP protokola

Sestavljena je iz dveh komponent:

- `upravitelj` poda agentu zahtevek za podatke in pravilno reagira v primeru prispelih dogodkov

- `agent` z zbranimi podatki odgovarja na zahteve upravitelja ter ga obvešča o dogodkih, ki so se zgodili na nadzorovanem sistemu [21]



Slika 2.4: Zgradba protokola CMIP[4].

2.2.2 Opis protokola

Ker morajo naprave podpirati celoten OSI model je uporaba protokola CMIP protokola precej redka. Zaradi tega pri nadzoru naprav preko interneta ali v lokalnih omrežjih prevladuje SNMP protokol, ki je enostavnejši za implementacijo in veliko bolj podprt s strani naprav povezanih v omrežje preko TCP/IP protokola.

Kej sej je uporaba SNMP protokola tako močno razširila in obstaja veliko število SNMP agentov, se je ta začela uporabljati tudi v namene nadzora TMN omrežij.

Tipi sporoči, ki definirajo prenos podatkov so[5]: M-CREATE, M-DELETE, M-GET, M-SET, M-ACTION, M-EVENT_REPORT.

Podatki, ki jih CMIP protokol prenaša so MIB podatki, definirani s SMI (*Structure of Management Information*). Struktura je zgrajena na enaki osnovi kot SNMP MIB, kar je razvidno tudi iz pomena naštetih tipov sporočil. Enaka osnova, pa v določenih primerih omogoča tudi pretvorbo iz SNMP v CMIP zapis [11].

[4]http://www.cellsoft.de/telecom/pics/manag2.gif
[5]ISO 8571-4 1/92 x.227 4/9

2.2.3 Primerjava s SNMP protokolom

Prednosti CMIP protokola:

- CMIP model je zgrajen na podlagi OSI modela in zato veliko natančneje sledi standardom
- ponuja boljšo varnost
- omogoča veliko večji nadzor podprtih naprav

Slabosti:

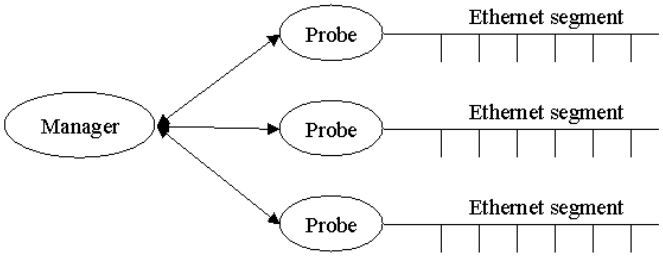
- potrebuje veliko več virov, kot SNMP in za to je potrebna zelo zmogljiva strojna oprema
- zgradba CMIP protokola je zelo kompleksna in zaradi velikega števila funkcionalnosti zahteva veliko predznanja in vloženega časa za implementacijo
- zaradi velike kompleksnosti je večje število možnosti za napake

2.3 RMON

RMON (*Remote Network Monitoring*) je sistem za nadzor in upravljanje omrežij. Sprva so bile RMON nadzorne postaje samostojne naprave oziroma kasneje razširitvene kartice za mrežno opremo, danes pa je RMON že programsko vgrajen v veliko število mrežnih komponent, predvsem za komercialno uporabo. Uporaba je možna tudi preko programske opreme na računalnikih in strežnikih [19]. Zaradi samostojnih enot namenjenih za nadzor omrežij, so podatki zelo uporabni, saj lahko točno določijo lokacijo v omrežju (vozlišča), kjer prihaja do preobremenitev ali napak in točen čas dogodkov.

2.3.1 Arhitektura RMON protokola

Za razliko od SNMP protokola, ki je centraliziran, vsak RMON agent (*probe*) skrbi za lokalni nadzor in nadzorni enoti (*manager*) posreduje analizirane podatke, s čimer zmanjša promet na omrežju.



Slika 2.5: RMON arhitektura[6].

V primeru nedosegljivosti nadzorne enot lahko RMON vseeno nadaljuje z zbiranjem podatkov in njihovo analizo. Nadzorni plošči jih posreduje takrat, ko ponovno postane dosegljiva.

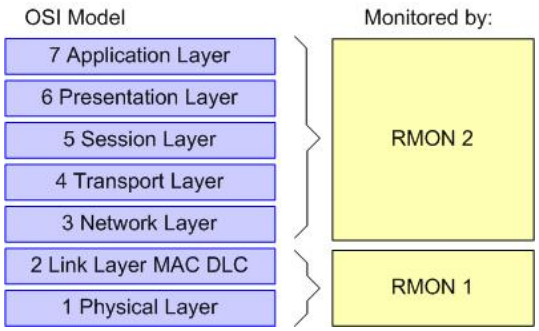
RMON ima definirana dva MIB standarda, RMON-1 in RMON-2 MIB [15].

2.3.2 RMON-1

RMON-1 standard je namenjen analiziranju povezovalne in fizične plasti ISO/OSI modela (vidno na sliki 2.6), filtriranju in zajemanju paketov ter proženju alarmov. Omogoča zajemanje naslednje skupine podatkov: *ethernet statistics*, *history control*, *ethernet history*, *alarm*, *host*, *hostTopN*, *matrix*, *filter*, *packet capture*, *event* [18].

[6]http://www.tml.tkk.fi/Opinnot/Tik-110.551/1999/papers/11ManagementOfPubIP/remote.gif

[7]http://www.loriotpro.com/Products/RMON_GUI/img/OSILayerRmon.jpg



Slika 2.6: Nadzor ISO/OSi modela z RMON-1 in RMON-2[7].

2.3.3 RMON-2 MIB

Kot omenjeno zgoraj, `RMON-1` podpira le analizo na prvih dveh plasteh `ISO/OSI` modela. `RMON-2 MIB` ne nadomešča prvotne verzije, ampak jo le razširja tako, da ima agent možnost nadzora in analize podatkov od povezovalne, pa vse do aplikacijske plasti ISO/OSI modela [16]. Prikaz je viden na sliki 2.6. Vsebovane skupine, implementirane v RMON-2 MIB, so: *the protocol directory group*, *protocol distribution group*, *address mapping group*, *network layer host group*, *network layer matrix group*, *application layer host group*, *application layer matrix*, *user history*, *probe configuration* [19].

2.4 Sklep

Po obravnavi protokolov SNMP, CMIP in RMON smo se odločili, da je protokol SNMP najbolj primeren za realizacijo naše rešitve. V primerjavi s protokolom CMIP, potrebuje veliko manj virov in zaradi preprostosti omogoča veliko lažjo in krajšo implementacijo. Prav tako ne potrebujemo tako obširnega nadzora, ki ga CMIP nudi. Od vseh obravnavanih protokolov je SNMP najbolj razširjen in podprt na mrežnih napravah, kot tudi s strani agentov.

Poglavje 3

Sistemi za upravljanje

S porastom uporabe interneta in spletnih tehnologij je nadzor lokalnih omrežij (LAN) postal nuja. Opisali smo že nekaj protokolov, ki so namenjeni nadzoru in upravljanju omrežij ter njihovih naprav, vendar za učinkovit nadzor potrebujemo sisteme, ki uporabljajo te protokole in uporabniku omogočajo lažji nadzor naprav, vpogled v stanje omrežja, vizualizacijo trendov in obveščanje v primeru napak.

3.1 Uvod

V tem poglavju bom predstavil tri primerljive odprtokodne sisteme, ki so namenjeni nadzoru razpoložljivosti omrežij, strežnikov, virtualnih sistemov, storitev, aplikacij in drugih mrežnih naprav ter njihovi zmogljivosti. Vsi v nadaljevanju opisani sistemi za pridobivanje podatkov od agentov podpirajo SNMP protokol, kar je glavna zahteva, saj smo ta protokol izbrali za prenos podatkov. Med seboj se razlikujejo v načinu shranjevanja podatkov in vizualizaciji podatkov preko spletnega vmesnika.

3.2 Zabbix

`Zabbix` je zelo zmogljivo orodje za nadzor trenutnega stanja agentov, ki preko spletnega vmesnika omogoča vizualizacijo trendov, grafov ter analiz zbranih podatkov. Skrbi tudi za obveščanje o anomalijah na nadzorovanih vozliščih oziroma ob akcijah nastavljenih sprožilcev. Ena od prednosti sistema pred konkurenčnimi odprtokodnimi sistemi, je zmogljiv spletni vmesnik, ki ga pogosto nadgrajujejo in s tem vse bolj izboljšujejo interakcijo med uporabnikom in vmesnikom.

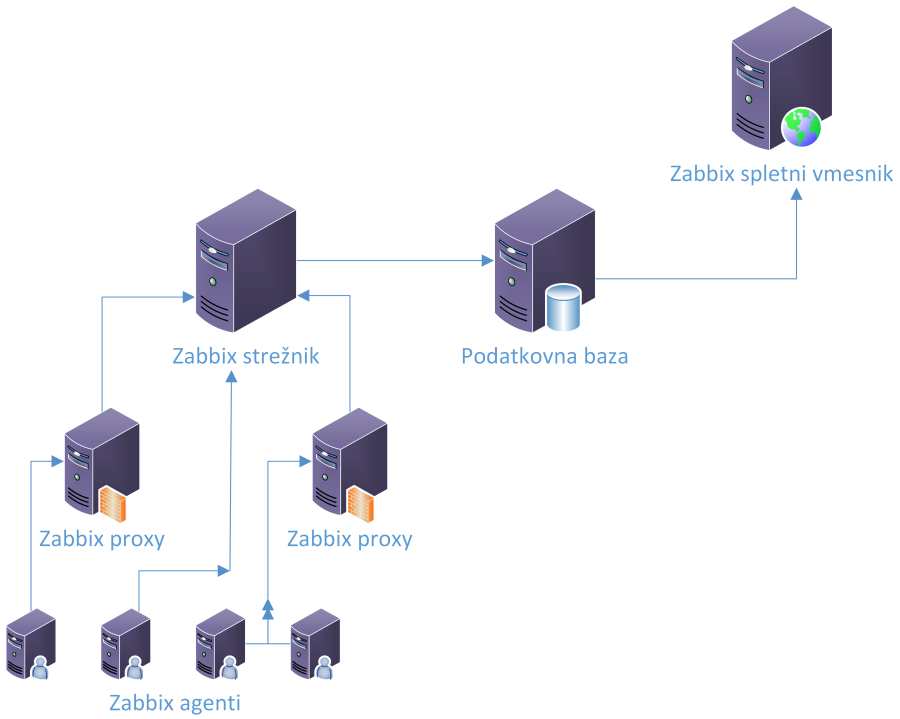
3.2.1 Arhitektura Zabbix sistema

Zabbix strežnik

Zabbix strežnik je centralna komponenta nadzornega sistema, ki od agentov in posredovalnih strežnikov prejema status razpoložljivosti in podatke o delovanju, kot je razvidno iz slike 3.1. Po obdelavi prejetih podatkov v primeru ugotovljenih napak ali sproženih alarmov, o tem obvesti uporabnike preko svojega obveščevalnega sistema. Uporabnik lahko nadzorni sistem nastavi in nadgradi po lastnih željah, kar je mogoče preko Zabbix API dostopa [12]. Zabbix server se lahko nahaja znotraj lokalnega sistema, ki ga nadzira, ali preko oddaljenega dostopa.

Podatkovna baza

Podatkovna baza vsebuje podatke o nastavitvah celotnega sistema, kot tudi podatke, ki jih obdela Zabbix strežnik. Pri manjših sistemih je baza nameščena ob Zabbix strežniku, vendar to ni pogoj. Podpira velik nabor relacijskih podatkovnih baz; MySQL, Oracle, PostgreSQL, SQLite, IBM DB2.



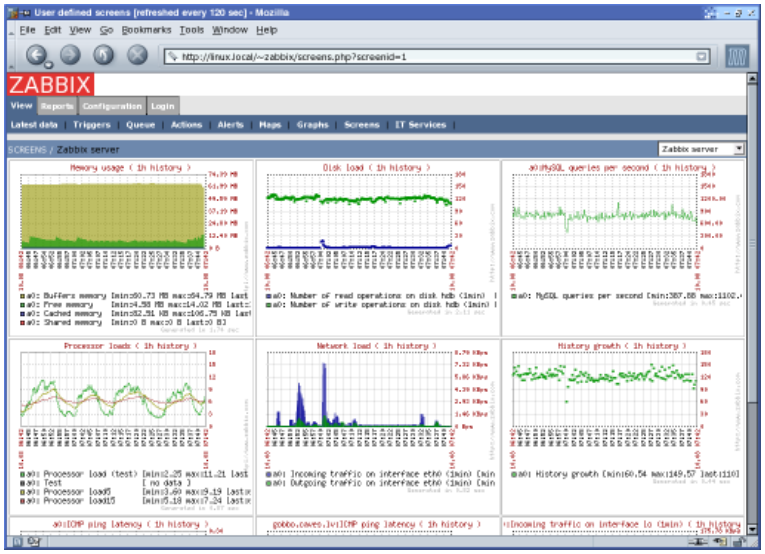
Slika 3.1: Zgradba Zabbix sistema.

Zabbix posredovalni strežnik

Zabbix posredovalni strežnik (*proxy*) lahko sprejema statuse in podatke enega ali večih agentov. Zabbix strežnik razbremeni, ta pa zato lahko več virov nameni obdelavi podatkov. Zabbix posredovalni strežnik potrebuje lokalno podatkovno bazo za vmesno shranjevanje podatkov.

Spletni vmesnik

Spletni vmesnik je del Zabbix strežnika, namenjen pregledu stanja vozlišč in vizualizaciji zbranih podatkov. Lahko je na istem vozlišču kot Zabbix strežnik ali pa na drugi lokaciji. Osnovan je na programskem jeziku PHP, ki za prikazovanje potrebuje Apache spletni strežnik. Primer je viden na sliki 3.2.



Slika 3.2: Primer spletnega vmesnika Zabbix[1].

Zabbix agent

`Zabbix agent` je nameščen na ciljni napravi, kjer nadzoruje lokalne vire in aplikacije ter zbrane podatke pošilja na Zabbix strežnik ali Zabbix posredovalni strežnik (*proxy*). Za pridobivanje informacij o vozlišču agenti uporabljajo sistemske klice, kar porablja minimalno virov. V primeru napak ali nedosegljivosti agenta, Zabbix strežnik poskrbi za poročilo o napaki in obveščanje uporabnikov.

Agent podpira dva načina pregledovanja:

- `pasiven` način zahteva minimalno viro, saj podatke posreduje Zabbix strežniku ali posredovalnemu strežniku le na zahtevo

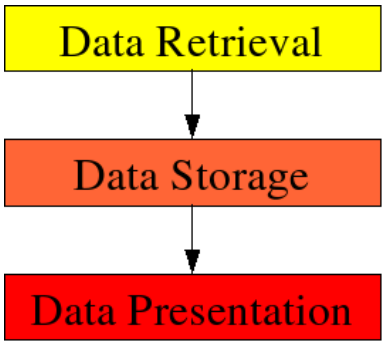
- `aktivno` preverjanje je zahtevnejše, najprej od Zabbix strežnika pridobi informacije za obdelavo, na podlagi katerih strežniku periodično pošilja podatke

[1]http://upload.wikimedia.org/wikipedia/commons/9/99/Zabbix.png

3.3 Cacti

Delovanje *Cacti* sistema lahko razdelimo na tri dele. Pridobivanje se izvaja na agentu, shranjevanje na upravitelju, prikazovanje pa je možno preko spletnega vmesnika [9].

3.3.1 Arhitektura sistema Cacti



Slika 3.3: Sestavni deli Cacti sistema[2].

Pridobivanje podatkov

Cacti poller je sistem, ki skrbi za pridobivanje podatkov od naprav. To doseže tako, da orodje *crontab* periodično pošilja zahtevke na naprave, ki podpirajo SNMP protokol. Naprava odgovori z zbranimi podatki o obremenitvi omrežja, centralni procesni enoti, vhodno izhodnih enotah, količini zavzetega pomnilnika in drugih virih.

Poleg privzetih virov lahko nadzorujemo tudi poljubne parametre, ki se pridobijo preko skript, ki so lahko napisane v *shell*, *perl*, *python* in *ruby* jeziku. Uporabniško prilagojene zahtevke lahko shranimo v predloge, ki jih distribuiramo po omrežju tudi na druge naprave. S temi funkcionalnostmi si lahko popolnoma prilagodimo nadzorovalni sistem.

[2]http://docs.cacti.net/_media/manual:087:principles_of_operation.png

Shranjevanje podatkov

RRDTool (*Round-robin database tool*) skrbi za shranjevanje in odelavo podatkov, njihovo hranjenje in generiranje grafov. Podatki se shranjujejo v *MySQL* bazo po principu krožnega pomnilnika (*Round-robin database*), kar ohranja podatkovno bazo konstantne velikosti, saj se najstarejši zapisi prepisujejo z novimi.

Spletni vmesnik

Spletni vmesnik omogoča prikaz najrazličnejših vrst grafov. V primeru, da uporabnik ni zadovoljen z osnovnim prikazom grafov, so ti nastavljivi in omogočajo shranjevanje v predloge. RRDTool, katerega naloga je med drugim tudi generiranje grafov, pa poskrbi za možnost uporabe predlog tudi na drugih sistemih.

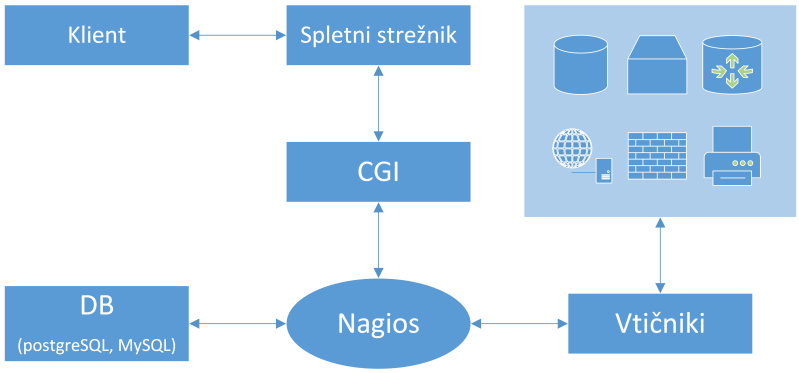
Nastavljivost in shranjevanje predlog je mogoča zaradi možnosti urejanja uporabniških računov in več nivojskih prioritet, za katere lahko uporabimo avtentikacijo uporabnikov, ki jo ponuja Cacti ali pa povezavo na izbrani LDAP strežnik.

3.4 Nagios

Tretji od obravnavanih nadzornih sistemov v diplomski nalogi je `Nagios`. Omogoča visok nivo skalabilnosti in prožnosti, kar je razlog za zelo pogosto uporabo v velikih organizacijah in podjetjih. Slabost sistema je zastarel uporabniški vmesnik in uporaba datotek za shranjevanje kompleksnih nastavitev, saj Nagios nima vgrajene podatkovne baze [6].

3.4.1 Arhitektura sistema Nagios

Arhitektura Nagios sistema je sestavljena iz jedra, vtičnikov, modulov, konfiguracije in uporabniškega vmesnika.



Slika 3.4: Arhitektura sistema Nagios.

Nagios jedro

Glavna naloga sistema je nadzor stanja naprav in storitev ter obveščanje v primeru napak. Samo jedro skrbi za razvrščanje nadzornih klicev, ki preverjajo statuse naprav in preko sporočilnega sistema obveščajo uporabnike v primeru okvar na omrežju ali zaznanih napakah.

Vtičniki

Vtičniki (*plug-ins*) so samostojne aplikacije, ki na podlagi podanih argumentov izvedejo zahtevane akcije in vrnejo rezultat. Nagios podpira dve vrsti vtičnikov.

- *check* vtičnik je namenjen preverjanju statusa nadzorovane naprave ali storitve. Na podlagi odgovora Nagios jedro primerno ukrepa
- *notification* vtičnik je namenjen posredovanju statusa in za obveščanje v primeru spremembe

Moduli

Moduli so dodatna programska oprema, ki preko Nagios API (*Application Programming Interface*) imenovanega *Nagios Event Broker* dostopa do podatkov v primeru določenega dogodka. S tem lahko registriramo metodo, ki

se bo začela izvajati ob določenem stanju dogodka. Na primer, če želimo ob padcu strežnika poslati SMS administratorju, lahko preko API dostopa registriramo metodo, ki se bo izvedla ob končani obdelavi dogodka, ki je rezultat padca naprave.

Konfiguracija

Kot že omenjeno, je ena od slabosti sistema shranjevanje konfiguracij v tekstovne datoteke. To povzroča težave na več nivojih; otežuje konfiguracijo za uporabnike, veča možnost napak ter manjša mobilnost konfiguracij med sistemi.

Spletni vmesnik

Kot ostala obravnavana sistema tudi Nagios ponuja spletni vmesnik za nadzor naprav in storitev, vendar je zastarelost vmesnika velika slabost. Spletni vmesnik je sprogramiran v `C` programskem jeziku, ki močno oteži integracijo novejših spletnih tehnologij, ki bi omogočale boljšo interakcijo med uporabnikom in sistemom. Ker je sistem odprtokoden, so se pojavile številne rešitve, s katerimi lahko nadomestimo privzet spletni vmesnik, vendar pa ne podpirajo vseh funkcionalnosti sistema.

3.4.2 Sklep

Vsi od obravnavanih sistemov podpirajo osnovne zahteve, kot so podpora SNMP protokola, nadzor omrežnih naprav in storitev, spletni vmesnik, ki omogoča vizualizacijo zbranih podatkov in uporabniško prilagoditev vmesnika.

Za Zabbix sistem smo se odločili, saj ponuja visoko stopnjo nastavljivosti, zelo dober spletni vmesnik z visoko stopnjo interaktivnosti in možnost serviranja preko spletnega strežnika Nginx. Ker je sistem zelo razširjen, je na voljo tudi veliko število predlog in agentov za naprave in vire, ki niso vključene v osnovni paket Zabbix sistema.

Poglavje 4

Visokorazpoložljive storitve

V zadnjem času smo priča vse večjemu številu storitev, ki svoje delovanje migrirajo na splet. To so na primer spletno bančništvo, naročilni sistemi za zdravstvo, B2B (*Business To Business*) poslovanja, poslovne aplikacije... S tem so se uporabniške zahteve zelo povečale in toleranca za nedosegljivost storitev drastično zmanjšala.

Toleranco za izpade merimo v odstotkih časa nedosegljivosti na leto [2]. Ta je definirana s številom "devetk"kot prikazuje tabela 4.1.

Dostopnost	Nedosegljivost na leto
99,9%	525.6 min ali 8.76 ur
99,99%	52.55 min
99,999%	5.25 min

Tabela 4.1: Visoka razpoložljivost v procentih

100% delovanja strojne in programske opreme v praksi ne moremo zagotoviti, lahko pa se s sistemi za preklop temu močno približamo. Za zagotavljanje visoke razpoložljivosti ni dovolj le podvajanje strojne in programske opreme. Potrebujemo tudi nadzorne sisteme, protokole za nadzor in izvajanje preklopa med vozlišči v primeru napak. Glavni namen je, da se uporabniku zagotovijo nemoteno uporabljanje storitve. V tem poglavju bom opisal principe in elemente, ki jih uporabljamo za zagotavljanje visoke razpoložljivosti.

4.1 Načini delovanja

Zgradbe visokorazpoložljivih sistemov se lahko med seboj zelo razlikujejo, saj ima vsak sistem drugačne zahteve. Določeni sistemi zadovoljujejo le minimalne zahteve visoke razpoložljivosti, druge pa poleg tega skrbijo še za porazdeljevanje virov.

Nekaj standardnih različic je opisanih v nadaljevanju [10]:

- `aktivno-aktivno`, pri tem načinu sta oba sistema aktivna in na njih teče enaka konfiguracija
- `aktivno-pasivno`, pri tem je en sistem nadrejen, drugi pa se aktivira ob padcu nadrejenega.
- `N-do-1`, začasno čakajoče vozlišče postane aktivno in ostane tako, dokler se primarno vozlišče ponovno ne vzpostavi
- `N-do-N`, kjer so vsa vozlišča aktivna. V primeru padca katerega od vozlišč, se obremenitev prerazporedi med ostala aktivna vozlišča. Ta konfiguracija ponavadi zahteva skupni dokumentni sistem

4.2 Pacemaker

Pacemaker je upravljalnik virov za gruče strežnikov (*clusters*), s katerim želimo doseči visokorazpoložljivost sistemov. Vsebuje sistem za detekcijo napak in povrnitev vozlišča v stanje delovanja ter zagotavlja nemoten preklop med vozlišči.

4.2.1 Arhitektura

Arhitektura je sestavljena iz treh glavnih komponent [1]:

- *non-cluster aware components* je komponenta upravljalnika, ki vsebuje vire (strojna oprema) in programsko opremo, ki ne vplivajo na delo-

vanje gruče, niti ne skrbijo za njegovo upravljanje. Na sliki 4.1, del označen z modro barvo

- upravljalec virov (*Resource management*) je komponenta, ki skrbi za obdelavo dogodkov in njihov odziv. Pod dogodke štejemo pridružitev ali izpad vozlišč zaradi napak, vzdrževalnih del ali administrativnih akcij. Po obdelavi dogodka bo ta komponenta poskušala vzpostaviti idealno stanje delovanja gruče strežnikov. Na sliki 4.1, del označen z zeleno barvo

- infrastruktura nižjega nivoja (*Low level infrastructure*) je komponenta, ki skrbi za najnižji nivo delovanja in je namenjena zaznavanju anomalij v delovanju ter na podlagi tega sproža dogodke. Za to Pacemaker uporablja sistem Corosync, saj nudi zanesljiv sporočilni sistem preko *unicast* ali *multicast* komunikacije in skrbi za zanesljivo komunikacijo med vozlišči, kot tudi za ponovni zagon aplikacij v primeru napak. Na sliki 4.1 del, označen z rdečo barvo

Lokalni upravljalec vira

Lokalni upravljalec vira (*Local Resource Manager*) upravlja s komponento *Resource Agents*, ki vsebuje tri različne tipe agentov.

- LSB agenti so zagonske skripte, ponavadi podane s strani operacijskega sistema, oziroma jih lahko napišemo sami, dokler ustrezajo LSB standardu

- OCF agenti so razširitve LSB agentov, ki poleg nadzora opravljajo še dodatne naloge. Ob namestitvi se nahajajo v paketu Pacemaker, vendar jih lahko dodajamo in spreminjamo, dokler ustrezajo zahtevam

[1] http://clusterlabs.org/doc/en-US/Pacemaker/1.0/html/Pacemaker_Explained/images/pcmk-overview.png

Slika 4.1: Pacemaker arhitektura[1].

- v literaturi in nastavitvah pogosto srečamo še *Heartbeat* agente, ki so malo spremenjena različica LSB agentov in so le še ohranjena podpora bivšega paketa Heartbeat, ki je podrobneje opisan v podpoglavju 4.4

Notranje komponente

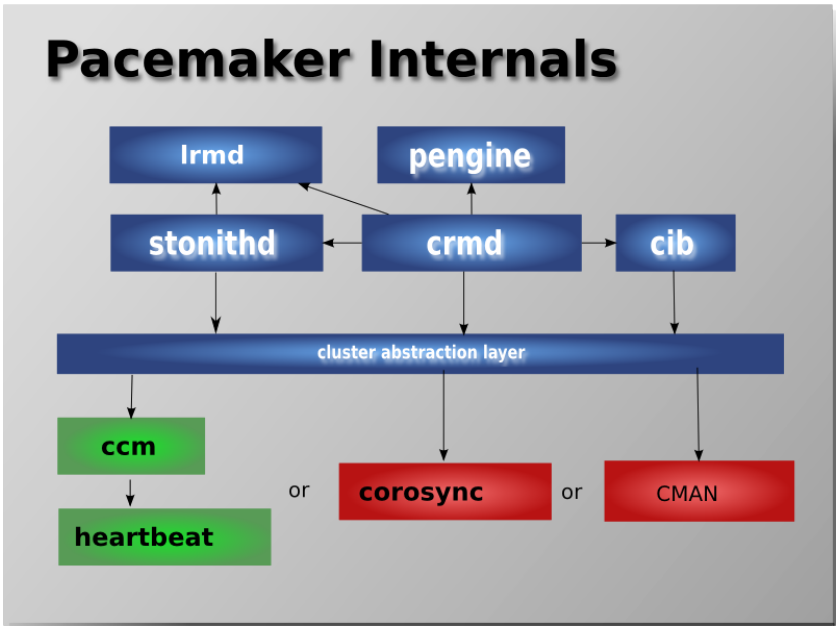
- *CIB (Cluster Information Base)* skrbi za samodejno usklajevanje vseh `XML` datotek, ki vsebujejo nastavitve in ažurno stanje virov v gruči.

- *PEngine (Policy Engine)* procesira podatke, ki jih priskrbi `CIB` in načrtuje, kako doseči idealno stanje. Ti podatki so posredovani `DC` (*Designated Controller*) enoti, ki eno od CRMd instanc postavi v nadrejen položaj. V primeru padca te komponente se hitro določi nova.

- *CRMd* (*Cluster Resource Management deamon*) obdeluje podatke, ki jih v določenem vrstnem redu dobi preko sporočilnega sistema od `DC` enote in posreduje naprej enoti `LRMd`. Vozlišča odgovarjajo nazaj DC

enoti.

- *LRMd (Local Resource Management deamon)* na lokalnem vozlišču izvrši pridobljene ukaze.
- *STONITHd* (*Shoot The Node In The Head*) upravlja z dovodom energije in po ukazu od LRMd ali CRMd izklopi vozlišče.



Slika 4.2: Shema notranjih komponent[2].

4.3 Corosync

`Corosync` (*The Corosync Cluster Engine*) je ogrodje namenjeno zagotavljanju visoke razpoložljivosti v kombinaciji z upravitelji visokorazpoložljivih sistemov, kot sta Pacemaker, *Apache Qpid* in drugi [4].

[2]http://clusterlabs.org/doc/en-US/Pacemaker/1.1-crmsh/html-single/Clusters_from_Scratch/images/pcmk-internals.png

Eden od glavnih ciljev Corosync ogrodja je zagotoviti čim manjšo tehnološko razdrobljenost, ki je problem primerljivih sistemov. Tako se pogosto zgodi, da sistemi informacije zajemajo nekonsistentno in tako vsako vozlišče posebej sprejema odločitve.

Corosync to težavo rešuje z deljenjem infrastrukture na osnovno infrastrukturo in na vse storitve, ki upravljajo z gručo. Tako lahko vsi strežniki znotraj gruče sodelujejo in doprinesejo informacije, na podlagi katerih se sprejema odločitve [14].

4.3.1 Zamenjava komponent med delovanjem

Celotno delovanje ogrodja Corosync je razdeljeno na komponente. Seznam komponent lahko bralec najde v članku [14]. Te so ob zagonu naložene statično ali dinamično. V primeru, da je naslovljena dinamično naložena komponenta, se ob naslovitvi prebere vmesnik iz notranjega pomnilnika. Če vmesnik ni bil najden, se pregleda poseben imenik, iz katerega se v primeru ujemanja dinamično naloži nov vmesnik. To funkcionalnost omogoča vtičnik *Live Component Replacement*, ki je uporabljen za zamenjavo komponent v času delovanja.

4.3.2 Pogon za usklajevanje

Kot že prej omenjeno, je ogrodje Corosync fragmentirano. Brez učinkovitega načina povezovanja teh komponent bi bilo ogrodje neuporabno in ravno za to skrbi komponenta *synchronization engine*. Uporabljen je tudi za vodenje postopkov, za obnovitev vozlišč v primeru napak ali ob vstavitvi dodatnega vozlišča.

Pri tem so mogoča štiri stanja:

- *sync_init* je prvi korak usklajevanja. Pri tem so shranjeni podatki o postopku za obnovitev.

- *sync_process* je korak, ki izvrši obnovitveni postopek.

- *sync_activate* je izvršen, ko je usklajevanje končano na vseh vozliščih.
- *sync_abort* se sproži, če se med usklajevanjem pridruži novo vozlišče ali, če kakšno preneha delovati. V tem primeru se obnovi stanje in ponovno sproži *sync_init*.

4.4 Heartbeat

Pojmovanje termina Heartbeat se je tekom razvoja visokorazpoložljivih sistemov nekajkrat spremenilo. V začetku je bil Heartbeat ime paketa, ki je vseboval celovito visokorazpoložljivo rešitev. Vsebovani paketi so bili sporočilni sistem, *Local Resource Manager*, *STONISH*, *Resource Agents* in *Cluster Resource Manager*. Kasneje se je paket razdelil in vsaka od omenjenih komponent je postala svoj paket. Določene komponente so danes del Corosync paketa, *Cluster Resource Manager* pa je poznan pod imenom Pacemaker.

Pomen, ki ga Heartbeat predstavlja danes, je izključno sporočilni nivo, ki se pogosto pojavlja tudi v današnjih sistemih, saj je osnova za veliko agentov, ki jih podpira Pacemaker.

4.5 Baza podatkov

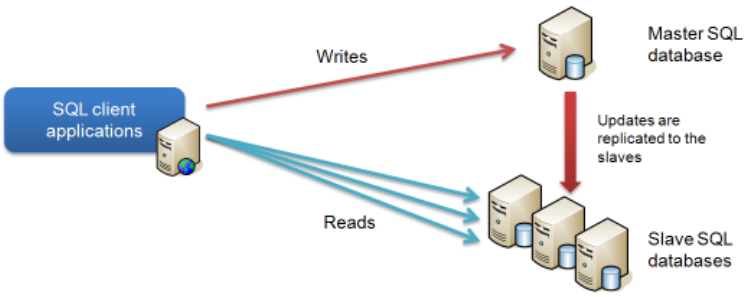
Poleg visoke razpoložljivosti same storitve je za nemoteno delovanje in preprečitev izgube podatkov potrebno zagotoviti tudi visoko zanesljivost podatkovne baze. Ker je med nastavitvami podatkovne baze in zahtevami aplikacijskega dela velika odvisnost, ne moremo za vse vrste arhitektur uporabljati le enega sistema, ki bi odgovarjal vsem različnim načinom uporabe.

4.5.1 Replikacija podatkovne baze

Replikacija podatkovne baze je proces kopiranja podatkov iz ene podatkovne baze na eno ali več kopij v porazdeljenem podatkovnem sistemu za zagotavljanje visoke dostopnosti [2].

Asinhrona replikacija

Asinhron način je kopiranje in uporaba transakcijskih zapisov WAL (*Write-Ahead Log*) na strežnikih znotraj porazdeljenega sistema. Pri tem načinu je podatek potrjen takoj, ko ga prejme lokalna baza. Sprememba je asinhrono prenešena na ostale sisteme z majhnim zamikom, kar lahko privede do potencialne izgube tistih podatkov, ki še niso bili porazdeljeni. Prednost tega je hitrost, saj za potrditev ni potrebno čakati ostalih instanc.

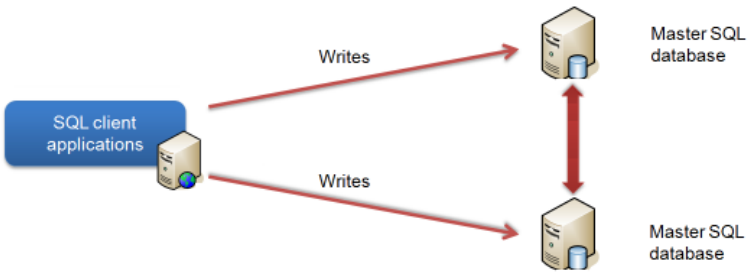


Slika 4.3: Shema asinhrone replikacije[3].

Sinhrona replikacija

Sinhron način uporablja 2PC (*Two-phase commit*) in zagotavlja potrditev ali razveljavitev transakcij na vseh instancah podatkovne baze naenkrat. Pri sinhroni replikaciji tako v primeru padca strežnika preprečimo izgubo podatkov (*zero data loss*).

[3]https://baoz.net/wp-content/HLIC/d120cc1230f5011f1e1afa2c44e8077c.png



Slika 4.4: Shema sinhrone replikacije[4].

Pri replikaciji pogosto srečamo tudi izraze `aktivno-aktivno` in `aktivno-pasivno` [5].

Aktivno-aktivno

Gre za dvosmerno replikacijo, kjer obe instanci podatkovne baze aktivno posodabljata podatke. Tu se promet porazdeljuje na oba strežnika. Ta način imenujemo tudi *multi-master replication*.

Aktivno-pasivno

Obratno kot pri `aktivno-aktivno` je tu replikacija enosmerna, kjer se iz aktivne podatkovne baze spremembe prenašajo na pasivni strežnik, ki je v stanju pripravljenosti. Ta način imenujemo tudi *master-slave replication*.

4.5.2 Zrcaljenje

Zrcaljenje (*mirroring*) služi za sprotno izdelovanje kopij podatkovne baze in ne kot način za implementacijo visokorazpoložljive rešitve, saj slednje ne zagotavlja. Pri zrcaljenju se izdeluje točna kopija, kar pomeni, da je baza

[4]Predelano po sliki: https://baoz.net/wp-content/HLIC/d120cc1230f5011f1e1afa2c44e8077c.png

posodobljena istočasno kot originalna baza in se zato uporablja za izvedbo sinhronizirane replikacije.

4.5.3 Streaming replication

Streaming replication je poseben način podvajanja podatkov pri podatkovni bazi PostgreSQL. Podatki, ki se pretakajo, so WAL zapisi, ki se iz aktivnega vozlišča prenašajo na ostala vozlišča v gruči. Omenjen način podvajanja ni omejen z določeno arhitekturno zgradbo, saj je lahko gruča nastavljena na delovanje po principu aktivno-aktivno ali aktivno-pasivno. Vsak od načinov pa je lahko sinhron ali asinhron.

Prednost pretakanja (*streaming replication*) je stanje pasivnega vozlišča, ki lahko deluje v načinu *hot_standy*, kar pomeni, da je vozlišče v stanju pripravljenosti in v primeru padca aktivnega vozlišča z zelo majhno zakasnitvijo omogoči zapisovalne poizvedbe in svoje stanje preklopi v aktivno.

Poglavje 5

Eksperimentalno vrednotenje

V poglavju 5 bomo predstavili realizirane rešitve in rezultate. Naš glavni namen je postaviti visokorazpoložljivo Zabbix storitev s kar se da malo porabljenimi sredstvi in ugotoviti, ali je le ta konkurenčna komercialnim rešitvam. V ta namen je potreben preklop, ki je sprejemljiv glede na uporabniške zahteve.

V nadaljevanju je podrobneje opisana arhitektura sistema ter vse komponente, ki povezane skupaj sestavljajo omenjeno rešitev. Vsi uporabljeni programski paketi in operacijski sistem so odprtokodni, zato njihova uporaba ne predstavlja finančnega stroška. Vseeno pa je potrebno upoštevati čas, ki je potreben za postavitev in nastavitev sistema, saj sistem brez dodatnega nastavljanja ne bo deloval takoj po namestitvi.

5.1 Arhitektura sistema

Visokorazpoložljive storitve so iz arhitekturnega stališča veliko bolj zahtevne in dražje kot storitve, ki tečejo na enem vozišču, saj za nemoteno delovanje v primeru preklopa potrebujemo vsaj dvojno število komponent. Danes srečamo v praksi velikokrat sisteme, ki tečejo na večjem številu vozlišč, kot bi bilo potrebno za zadostitev osnovnih pogojev visoke razpoložljivosti, vendar je pri tem visoka razpoložljivost združena z razporejanjem obremenitve (*load*

balancing), ki pa v osnovi nista soodvisni.

Zaradi lažje preglednosti in enostavnosti rešitve bo Zabbix sistem tekel na dveh vozliščih s 64 bitnim operacijskim sistemom `Ubuntu 14.04 LTS`. Vsako vozlišče ima svoj statičen IP, vendar bomo poleg tega potrebovali še tri navidezne IP naslove.

V kodi 5.1 za boljše razumevanje in lažje nastavljanje na vseh vozliščih nastavimo datoteko `/etc/hosts`, ki bo vsebovala privzeta imena (*Fully qualified name*), kar nam bo omogočilo uporabo privzetih imen namesto IP naslovov.

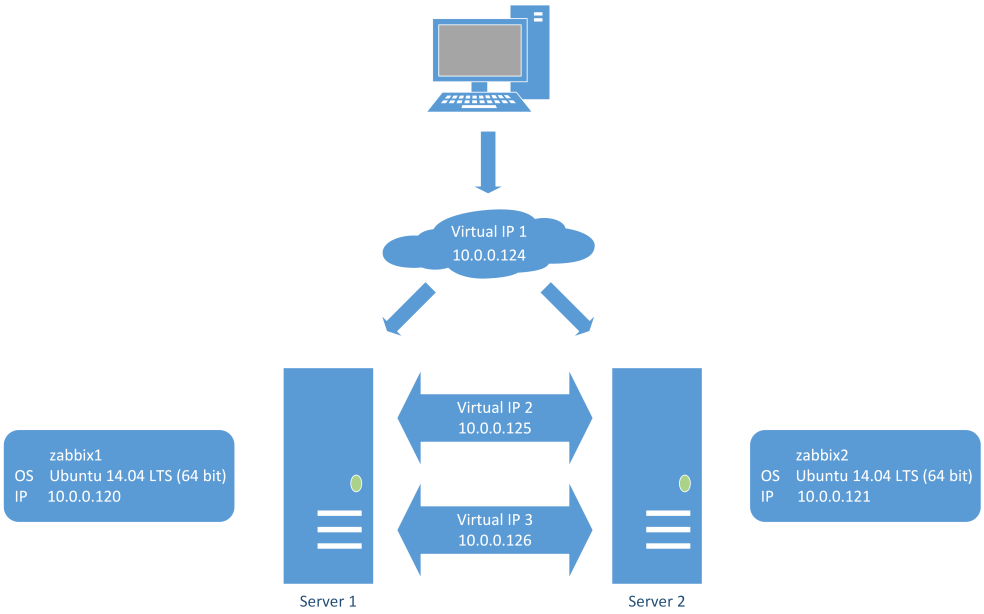
Koda 5.1: Nastavitev privzetih imen

```
10.0.0.120    zabbix1
10.0.0.121    zabbix2
10.0.0.124    zabbix
10.0.0.125    pgmaster
10.0.0.126    pgreplic
```

Navidezni IP 10.0.0.124 bo uporabljen za dostop do spletnega vmesnika Zabbix, naslova 10.0.0.125 in 10.0.0.126 pa za povezavo na aktivno vozlišče podatkovne baze in povezavo za podvajanje.

Vsi ukazi, ki jih bomo izvrševali, morajo biti izvršeni s pravicami korenskega uporabnika (sudo), ali pa morajo spadati v skupino s pravilno nastavljenimi pravicami.

IP naslovov nam ni potrebno nastavljati ročno, saj za to nastavljanje in menjavo skrbi `Pacemaker`, kar je podrobneje opisano v podpoglavju 5.5.



Slika 5.1: Arhitektura visokorazpoložljivega sistema.

5.2 Podvajanje podatkovne baze PostgreSQL

Pred začetkom namestitve Zabbix sistema moramo namestiti in nastaviti podatkovno bazo. Ker je namen diplomske naloge postaviti visokorazpoložljivo Zabbix storitev, je treba poskrbeti tudi za podvajanje in preklop podatkovne baze. V nadaljevanju se bomo posvetili podvajanju, preklopu pa je namenjeno podpoglavje 5.5.3.

Koda 5.2: Namestitev in nastavitev podvajanja PostgreSQL podatkovne baze

```
apt-get install postgresql-9.3
mkdir /opt/pg_archive

# nastavitve v datoteki postgresql.conf
listen_addresses = '*'
wal_level = hot_standby
synchronous_commit = on
```

```
archive_mode = on
archive_command = 'cp %p /opt/pg_archive/%f'
max_wal_senders=5
wal_keep_segments = 70
hot_standby = on
hot_standby_feedback = on
restart_after_crash = off

# nastavitve v datoteki pg_hba.conf
host  replication  all  10.0.0.0/24  trust
host  zabbixdb     all  10.0.0.0/24  md5
```

Nastavitve iz kode 5.2 moramo nastaviti na obeh vozliščih. Najprej namestimo PostgreSQL podatkovno bazo in naredimo mapo, kamor se bodo arhivirale WAL datoteke. Ker želimo nastaviti podvajanje tipa *Streaming replication*, ki se poslužuje *hot standby* načina, moramo parameter *wal_level* nastaviti na *hot_standby*. Tako se bodo WAL datoteke prenašale z zelo malo zakasnitve in v primeru preklopa bo pasivno vozlišče hitro omogočilo sprejem zapisovalnih poizvedb ter prevzelo vlogo aktivnega vozlišča.

V datoteki `pg_hba.conf` je treba nastaviti pravice, ki omogočajo dostop do podatkov in podvajanje. Pomembno je, da pri vnosu za podvajanje namesto imena podatkovne baze nastavimo *replication*. V našem primeru smo dostop omejili na podmrežje 10.0.0.0 in zaščito pri podvajanju nastavili na *trust*, kar ni priporočljivo za produkcijo, vendar nam olajša delo v testnem okolju.

Ob začetku podvajanja potrebujemo na pasivnem strežniku datoteko *recovery.conf*, ki se mora nahajati v imeniku, kamor se shranjujejo podatki podatkovne baze.

Koda 5.3: Nastavitve recovery.conf datoteke

```
standby_mode = 'on'
primary_conninfo = 'host=10.0.0.126 port=5432 \
    user=postgres application_name=zabbix2'
restore_command = 'cp /opt/pg_archive/%f %p'
recovery_target_timeline = 'latest'
```

Omenjene datoteke, katere vsebino lahko vidimo v kodi 5.3, kasneje v našem primeru ne bomo potrebovali, saj jo bo sistem za zagotavljanje visoke razpoložljivosti nadomestil s svojimi nastavitvami.

Koda 5.4: Testiranje uspešnega podvajanja

```
psql
  select client_name, sync_state from pg_stat_replication;
    client_name  | sync_state
    -------------+------------
    zabbix2      | async
```

V kodi 5.4 lahko po končanem nastavljanju preverimo, ali je podvajanje uspešno. Kot lahko vidimo, je v našem primeru nastavljeno asinhrono podvajanje iz vozlišča `zabbix1` na vozlišče `zabbix2`.

5.3 Namestitev Zabbix sistema

Zabbix je odprtokoden sistem, katerega prenos je na voljo brezplačno preko uradne spletne strani[1]. Pred začetkom namestitve smo na obe vozlišči prenesli zadnjo stabilno verzijo izvorne kode (`Zabbix 2.2.5`).

Ker sem sistem namestili preko izvorne kode, je bilo pred postavitvijo sistema namestiti vse zahtevane pakete ter v operacijskem sistemu ustvariti novega uporabnika in skupino, kot vidimo v kodi 5.5.

Koda 5.5: Ustvarimo nov uporabniški račun in skupino

```
groupadd zabbix
useradd -g zabbix zabbix
```

Sledilo je nameščanje vseh zahtevanih paketov, ki so potrebni za pravilno postavitev s parametri, ki bodo opisani v nadaljevanju.

Koda 5.6: Namestitev zahtevanih paketov s strani Zabbix sistema

```
apt-get install gcc
apt-get install make
```

[1] http://www.zabbix.com/

```
apt-get install libpq-dev
apt-get install libxml2-dev
apt-get install libsnmp-dev

apt-get install oracle-java7-set-default
```

Po namestitvi zahtevanih paketov je sledila konfiguracija in namestitev Zabbix sistema, kot je zapisano v kodi 5.7

Koda 5.7: Nastavitev in namestitev sistema Zabbix

```
./configure --enable-server --enable-agent
            --enable-java --with-postgresql
            --with-libxml2 --with-net-snmp
make install
```

Parametra `--enable-server` in `--enable-agent` omogočata delovanja Zabbix sistema v strežniškem in agent načinu. Strežniški način je ključnega pomena pri razvoju rešitve, agent način pa bomo uporabili za nadzor dotičnega strežnika, predvsem iz testnih razlogov. Parameter `--enable-java` omogoči preko `Zabbix Java Gateway` demona nadzor `JMX Java` aplikacij v primeru, da so bile zagnane z opcijo `-Dcom.sun.management.jmxremote`.

Pred zagonom strežnika moramo nastaviti še nekaj parametrov za pravilno delovanje strežnika in agenta. Koda 5.8 prikazuje parametre, ki jih je treba nastaviti v datotekah na obeh vozliščih uporabljenih v sistemu.

Koda 5.8: Nastavljanje Zabbix sistema

```
# Nastavljanje datoteke zabbix_server.conf
# Na strezniku zabbix1 in zabbix2
DBHost=10.0.0.125
DBName=zabbixdb
DBUser=zabbixdb
DBPassword=<geslo>

# Nastavljanje datoteke zabbix_agentd.conf
# Na strezniku zabbix1 in zabbix2
```

```
Server=zabbix1,zabbix2

# na strezniku zabbix1
Hostname=zabbix1

# na strezniku zabbix2
Hostname=zabbix2
```

5.4 Postavitev spletnega vmesnika

Ena od glavnih funkcionalnosti Zabbix sistema je interaktiven spletni vmesnik, ki omogoča tako administracijo in nastavljanje sistema, kot tudi obveščanje in vizualizacijo. Ker je napisan v programskem jeziku PHP, je za uporabo potreben spletni strežnik, ki omogoča serviranje PHP spletnih strani. Na voljo je kar nekaj neplačljivih paketov, kot na primer Apache, Nginx, lighttpd.

5.4.1 Namestitev Nginx spletnega strežnika

V Zabbix dokumentaciji je veliko navodil za nastavitev Apache strežnika, vendar smo se odločili za Nginx spletni strežnik. Slednjega bomo uporabili zaradi visoke zmogljivosti in možnosti uporabe php5-fpm modula, ki med drugim ponuja tudi dinamično število procesov, ki se prilagajo glede na obremenitev strežnika. Namestitev je vidna v kodi 5.9.

Koda 5.9: Namestitev Nginx spletnega strežnika

```
apt-get install php5-fpm
apt-get install php5-pgsql
apt-get install nginx
```

5.4.2 Nastavitev Zabbix sistema

Po namestitvi spletnega strežnika na obe vozlišči, moramo spletni strežnik še pravilno nastaviti. Koda je na voljo v kodi 5.10.

Ustvarimo mapo, ji nastavimo lastnika in skupino na `www-data` ter vanjo skopiramo kodo spletnega vmesnika, ki se nahaja v mapi Zabbix izvorne kode na lokaciji `/zabbix-2.2.5/frontend/php/`. Ker ob koncu namestitve Zabbix spletnega vmesnika nastavitve shranimo v datoteko, je treba pred tem še pravilno nastaviti pravice datoteki `zabbix.conf.php`, ki se nahaja v mapi `conf/`.

Koda 5.10: Nastavitev Nginx spletnega strežnika

```
mkdir /var/www/zabbix
cd ~/zabbix-2.2.5/frontend/php
cp * /var/www/zabbix/
cd /var/www/
chown -R www-data:www-data zabbix/
chmod 775 zabbix/conf/zabbix.conf.php
```

Pred zagonom spletnega strežnika je treba nastaviti in omogočiti dostop do spletne strani. V ta namen ustvarimo datoteko `zabbix.conf` v mapi `/etc/nginx/sites-available`.

Koda 5.11: Nastavitev dostopa do spletne strani

```
server {
  listen 80 default_server;
  server_name 10.0.0.124;
  root /var/www/zabbix;

  location / {
    index.php index.html index.htm;
    try_files $uri $uri/ =404;
  }

  location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass unix:/var/run/php5-fpm.sock;
    fstcgi_param SCRIPT_FILENAME
      /scripts$fastcgi_script_name;
    fastcgi_index index.php;
    include fastcgi_params;
```

```
    }
}

cd /etc/nginx/sites-enabled
ln -s ../sites-available/zabbix.conf .

service nginx start
```

Zabbix spletni vmesnik ima še nekaj dodatnih zahtev, ki jih je treba spremeniti v `php.ini` datoteki. Ker sem uporabil Nginx spletni strežnik s `php-fpm` podporo, ni potrebno spreminjati `php.ini` datoteke, saj bi te nastavitve vplivale na celoten sistem. Nastavitve lahko zapišemo v `php-fpm` konfiguracijsko datoteko `/php5/fpm/pool.d/www.conf` zabbix podstrani, kot je prikazano v kodi 5.12.

Koda 5.12: Nastavitev PHP konstant

```
php_admin_value[memory_limit] = 128M
php_admin_value[max_execution_time] = 300
php_admin_value[max_input_time] = 300
php_admin_value[post_max_size] = 16M
php_admin_value[date.timezone] = "Europe/Ljubljana"
```

5.4.3 Nastavitev podatkovne baze

Struktura podatkovne baze in vsebina, ki je potrebna za delovanje *Zabbix* spletnega vmesnika, sta na voljo v paketu izvorne kode. Treba je uvoziti shemo in podatke, kakor je prikazano v kodi 5.13.

Koda 5.13: Nastavitev podatkovne baze PostgreSQL

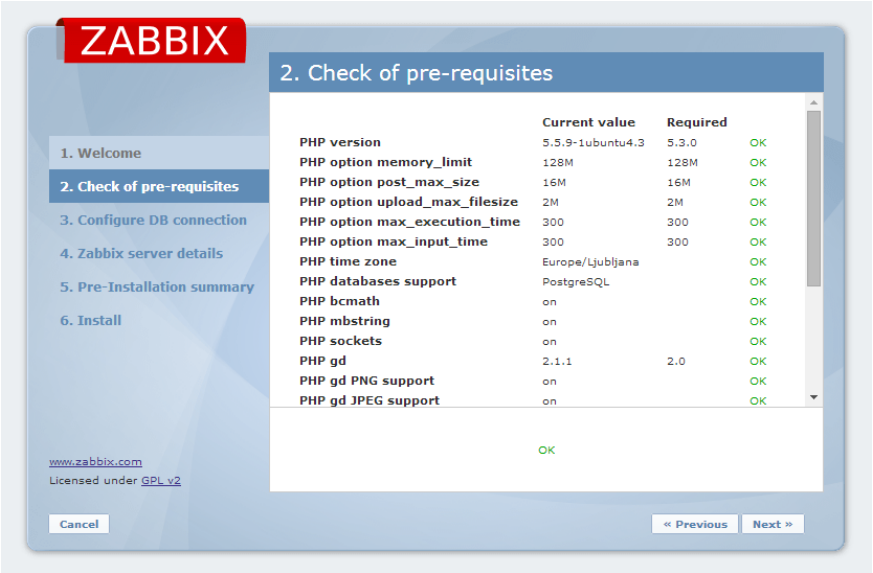
```
psql -U postgresql
  CREATE USER zabbixdb WITH PASSWORD '<geslo>';
  CREATE DATABASE zabbixdb;
  GRANT ALL PRIVILEGES ON DATABASE zabbixdb TO zabbixdb;

cd ~/zabbix-2.2.5/database/postgresql
psql -U zabbixdb zabbixdb -W < schema.sql
```

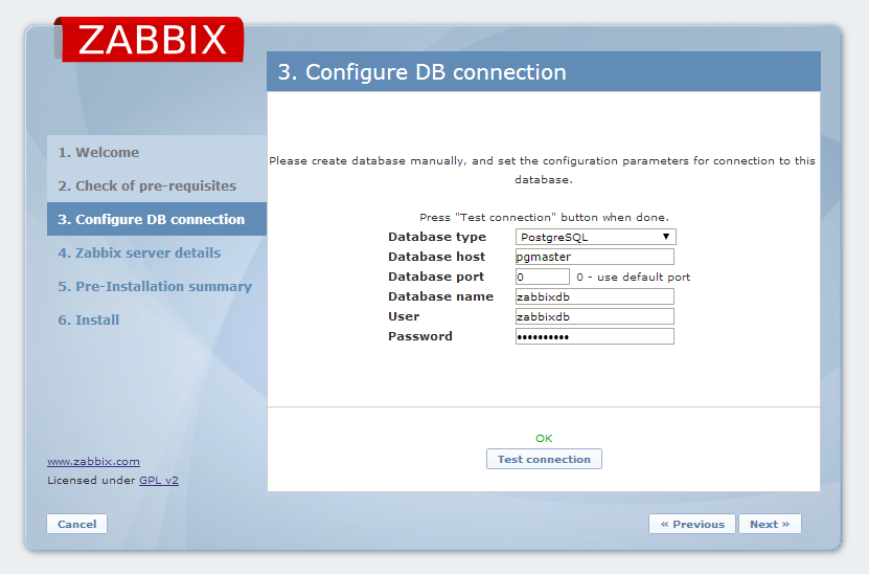
```
psql -U zabbixdb zabbixdb -W < images.sql
psql -U zabbixdb zabbixdb -W < data.sql
```

5.4.4 Nastavitev spletnega vmesnika

Nastavljeni so vsi potrebni elementi, da lahko v brskalniku odpremo spletni vmesnik in vnesemo še zadnje nastavitve za povezavo z bazo. Kot je vidno na sliki 5.2, so vsi PHP parametri nastavljeni pravilno. V naslednjem koraku (slika 5.3) je treba vnesti še podatke za dostop do podatkovne baze, ki so enaki tistim, ki smo jih vnesli pri konfiguraciji Zabbix strežnika v kodi 5.8.

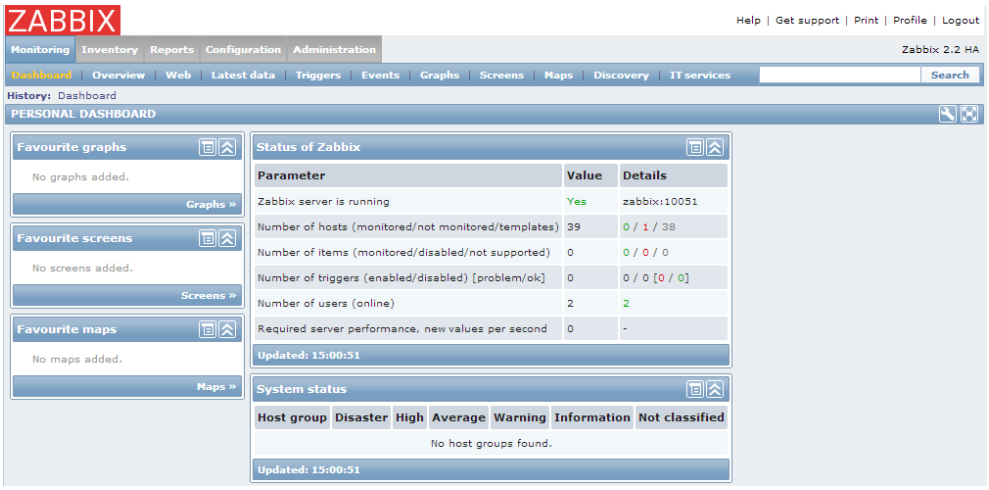


Slika 5.2: Pregled PHP konstant.



Slika 5.3: Povezava s podatkovno bazo.

Vnesemo še privzeto ime `zabbix`, preko katerega bo dostopen Zabbix strežnik. Enak postopek je treba ponoviti na obeh strežnikih. Po uspešni nastavitvi prvič vidimo Zabbix nadzorno ploščo (slika 5.4).



Slika 5.4: Zabbix nadzorna plošča.

5.5 Visoka razpoložljivost

Sistem Zabbix je uspešno naložen na obeh strežnikih, vendar manjka logika, ki bo omogočala preklop med njima v primeru napake na sistemu oz. nedosegljivosti. V teoretičnem delu sem omenil paketa Pacemaker in Corosync, ki se pogosto uporabljata za implementacijo visokorazpoložljivostnih rešitev. Ker je na vozliščih naložen Ubuntu operacijski sistem, je mogoče paketa naložiti preko sistema za upravljanje s paketi APT. Postopek ponovimo na obeh vozliščih.

Koda 5.14: Namestitev paketov Corosync in Pacemaker

```
apt-get install pacemaker corosync
```

5.5.1 Nastavitev paketa Corosync

Ker želimo, da so sporočila, ki jih Corosync pošilja med vozlišči avtenticirana, ponuja Corosync knjižnico *corosync-keygen*, s katero zgeneriramo ključ, ki ga je treba prenesti na vsa vozlišča v gruči. Celoten postopek je prikazan v kodi 5.15.

Koda 5.15: Generiranje Corosync avtentikacijskega ključa

```
corosync-keygen

scp /etc/corosync/authkey ubuntu@zabbix2:/home/ubuntu
mv ~/authkey /etc/corosync
```

Corosync smo namestili, vendar moramo pred zagonom nastaviti njegovo delovanje. To je potrebno narediti na vseh vozliščih, saj med tem definiramo naslov kanala, preko katerega bo vozlišče komuniciralo z ostalimi vozlišči v gruči. Definiramo še beleženje in druge parametre. Pisanje nove konfiguracijske datoteke bi bilo nesmiselno, zato je v paketu na lokaciji `/etc/corosync/` že pripravljen primer `corosync.conf.example`, ki ga lahko skopiramo in preimenujemo v `corosync.conf`, primer katerega lahko vidimo v kodi 5.16.

Zelo pomembna sta parametra `bindnetaddr` in `mcastaddr`. Prvega nastavimo na naslov podomrežja, kar omogoča kopiranje datoteke na ostala vozlišča, saj tako ne definiramo dotičnega vozlišča in nam ni treba v vsako datoteko navesti seznama naslovov vozlišč v gruči. Drugi parameter, `mcastaddr` je `multicast` naslov, ki določa skupino prejemnih naprav v omrežju. Slednjega lahko preberemo iz omrežnih nastavitev vozlišča z ukazom `netstat -g`.

Koda 5.16: Nastavljanje datoteke corosync.conf

```
...
interface {
    ringnumber: 0
    bindnetaddr: 10.0.0.0
    mcastaddr: 239.255.1.1
    mcastport: 5405
}
...
```

Za nemoteno delovanje je ob zagonu operacijskega sistema potreben zagon Corosynca, kar je privzeto izključeno. V datoteki `/etc/default/corosync` je treba parameter `START` spremeniti na `yes`.

5.5.2 Namestitev paketa Pacemaker

Pacemaker za razliko od paketa Corosync ne potrebuje dodatnega nastavljanja. Glavni del paketa Pacemaker so skripte, imenovane *resource agents*, ki so namenjene nadzoru in preklopu raznovrstnih storitev. V primeru, da ta ne vsebuje primerne skripte, lahko ob upoštevanju smernic napišemo svojo in jo dodamo v množico.

Ker paket, ki je na voljo, ne vsebuje zadnjih verzij agentov, sem te prenesel in namestil iz uradnega *ClusterLabs git* repozitorija[2]. Zdaj, ko imamo vse potrebne pakete in nastavitve, lahko preverimo stanje gruče, kot kaže koda 5.17.

[2]https://github.com/ClusterLabs/resource-agents

Koda 5.17: Stanje strežnikov v gruči

```
crm_mon -1
============
Last updated: Sat Set 6 21:30:27 2014
Stack: corosync
Current DC: zabbix1 (1) - partition with quorum
Version: 1.1.10-42f2063
2 Nodes configured
0 Resources configured
============

Online: [ zabbix1 zabbix2]
```

5.5.3 PostgreSQL

V podpoglavju 5.2 smo opisali namestitev in nastavitev podatkovne baze PostgreSQL, ki deluje v načinu podvajanja (*streaming replication*). Podvajanje podatkov iz aktivnega na pasivno vozlišče že poteka, vendar v primeru padca aktivnega strežnika ne pride do preklopa.

Pred nastavitvijo agentov za upravljanje preklopa moramo nastaviti navidezna IP naslova, ki bosta uporabljena za dostop do podatkovne baze in za podvajanje. V ta namen uporabimo *OCF (Open Cluster Framework)* agenta, ki je še del starega paketa Heartbeat. Gre za razširitev nadzornih linux agentov, ki so namenjeni za nadzor najrazličnejših virov, kot so IP naslovi, datotečni sistemi, podatkovne baze in drugi.

Koda 5.18: Nastavitev virtualnih IP naslovov podatkovne baze

```
crm configure edit

primitive vip-master ocf:heartbeat:IPaddr2 \
  params ip="10.0.0.125" cidr_netmask="24" \
    iflabel="1" nic="br-lan"
  op start timeout="60s" interval="0s" on-fail="stop"
  op stop timeout="60s" interval="0s" on-fail="block"
  op monitor timeout="60s" interval="10s" on-fail="restart"
```

Iz kode 5.18 je razvidno, da definiramo navidezni IP 10.0.0.125, kateremu določimo akcije za nadzor, zagon in izklop. Enak postopek ponovimo še za IP 10.0.0.126, kateremu določimo ime *vip-rep*. Pozorni moramo biti tudi na nastavitev imena mrežnega vmesnika, na podlagi katerega se nastavi navidezni vmesnik.

Naslednji korak je definicija nastavitev, ki bodo nadzirale podatkovno bazo.

Koda 5.19: Nastavitev preklopa PostgreSQL podatkovne baze

```
crm configure edit

primitive pgsql ocf:heartbeat:pgsql \
params \
  pgctl="/usr/lib/postgresql/9.3/bin/pg_ctl"
  psql="/usr/lib/postgresql/9.3/bin/psql"
  pgdata="/opt/pg_data/" node_list="zabbix1 zabbix2"
  rep_mode="async" master_ip="10.0.0.126"
  start_opt="-p 5432"
  restore_command="cp /opt/pg_archive/%f %p"
  primary_conninfo_opt="keepalives_idle=60 \
    keepalives_interval=5 keepalives_count=5"
  tmpdir="/var/run/postgresql" restart_on_promote="true"
  config="/etc/postgresql/9.3/main/postgresql.conf"
  op start interval="0s" timeout="60s" on-fail="restart"
  op promote interval="0s" timeout="60s" on-fail="restart"
  op demote interval="0s" timeout="60s" on-fail="stop"
  op stop interval="0s" timeout="60s" on-fail="block"
  op notify interval="0s" timeout="0s"
  op monitor interval="3s" role="Master" timeout="60s" \
    on-fail="restart"
  op monitor interval="4s" timeout="60s" on-fail="restart"
```

V kodi 5.19 uporabimo obstoječega *ocf resource* agenta `pgsql`, ki ima vgrajeno podporo za preklop. Določiti moramo pot do *pgctl* skripte, ki upravlja s PostgreSQL procesom, pot do vmesnika (*psql*) za upravljanje s podatkovno bazo ter pot do imenika, kjer se nahaja nastavitvena datoteka, kamor

se bodo shranjevali podatki. Agent podpira še veliko drugih parametrov, ki so potrebni za delovanje, kot na primer *node_list*, ki mora vsebovati imena vseh vozlišč, *master_ip* in pa *restore_command* ter *primery_conninfo_opt*, ki sta namenjena nastavitvi *recovery.conf* datoteke. Sledijo še parametri, ki skrbijo za operacije nad procesom, kjer določimo interval preverjanja in časovno omejitev.

Nastavitve je treba združiti v skupino tako, da se bodo viri preklapljali skupaj in preprečili nepravilno delovanje, do katerega bi prišlo, če bi bili virtualni IP naslovi nastavljeni na enem vozlišču, aktivni strežnik podatkovne baze pa na drugem. Za tem omejimo še število aktivnih in pasivnih strežnikov ter nastavimo vrstni red zagona virov. Ta je zelo pomemben, ker mora biti podatkovna baza pripravljena na poizvedbe, še preden jih strežnik začne sprejemati. Nastavitve so vidne v kodi 5.20.

Koda 5.20: Združevanje in nastavitev vrstnega reda zagona virov

```
group master-group vip-master vip-rep
ms msPostgresql pgsql meta master-max="1" clone-max="2" \
    master-node-max="1" clone-node-max="1" notify="true"
colocation rsc_colocation-1 inf: master-group \
    msPostgresql:Master
order rsc_order-1 0: msPostgresql:promote \
    master-group:start symmetrical=false
order rsc_order-2 0: msPostgresql:demote \
    master-group:stop symmetrical=false
```

5.5.4 Zabbix sistem

Za nastavitev preklopa uporabimo *LSB resource agents (Linux Standard Base)* agente. To so agenti, ki operirajo s skriptami, ki lahko zaženejo ali ugasnejo proces ter vračajo status delovanja. V osnovi gre za zagonske skripte, ki jih po navadi pridobimo od operacijskega sistema oziroma skupaj s programom ali pa napišemo svoje. Edini pogoj je, da zadoščajo `LSB` zahtevam.

Namestitev Zabbix sistema sem opisal že v podpoglavju 5.3, vendar za

samo delovanje ne potrebujemo visoke razpoložljivosti. Pred nastavitvijo potrebujemo skripte, ki skrbijo za zagon, izklop in vračilo statusa programa. Ker te skripte niso nameščene ob namestitvi Zabbix sistema iz izvorne kode, moramo zanje poskrbeti sami. Skripte z osnovnimi funkcionalnostmi se nahajajo v izvorni kodi, vendar je potrebno sprogramirati funkcionalnost za vračilo statusa, kar je vidno v kodi 5.21.

Koda 5.21: Nastavitev zagonskih skript Zabbix sistema

```
cp misc/init.d/debian/zabbix-server /etc/init.d

# Doplonitev datoteke /etc/init.d/zabbix-server
. /lib/lsb/init-functions
...
status|monitor)
    status_of_proc -p $PID $DAEMON $NAME \
        && exit 0 || exit $?
    ;;
...

chmod 755 /etc/init.d/zabbix-server
update-rc.d zabbix-server defaults

chmod 755 /etc/init.d/zabbix-agent
supdate-rc.d zabbix-agent defaults
```

V kodi 5.17 vidimo, da sta v gruči aktivni dve vozlišči, vendar ni podanih nobenih nastavitev. Zagonske skripte, ki smo jih nastavili v kodi 5.21, igrajo ključni pomen pri nadzoru aktivnosti Zabbix strežnika. V kodi 5.22 nastavimo pregledovanje aktivnosti procesa na 5 sekund. Poleg preverjanja stanja sistema dodamo še preverjanje dostopnosti privzetega imena `zabbix`.

Koda 5.22: Nastavitve nadzora Zabbix strežnika in aktivnosti navideznega IP naslova 1

```
crm configure edit

primitive ha-zabbix_server lsb::zabbix_server \
```

```
        op monitor interval="5s"
    primitive ha-ip ocf:heartbeat:IPaddr \
        params monitor="10.0.0.124" op monitor interval="2s"
    group ha-zabbix ha-zabbix_server ha-ip
    property ...
        default-resource-stickiness="100"
```

Oba ukaza za preverjanje združimo v skupino `ha-zabbix`, s čimer določimo, da se bo izvajanje premikalo z vozlišča na vozlišče skupaj. Tako preprečimo, da je IP aktiven na enem vozlišču, Zabbix strežnik pa teče na drugem. Na koncu dodamo še parameter `default-resource-stickiness`, s katerim kar se da omejimo preklapljanje med vozlišči. Tako se bo izvajanje preklopilo drugam le v primeru, da Pacemaker najde bolj optimalno vozlišče.

5.6 Vrednotenje

V poglavju 5 smo opisali vse postopke nalaganja in nastavljanja uporabljenih paketov ter preverili njihovo stanje. V nadaljevanju je viden test delovanja preklopa Zabbix sistema in podatkovne baze.

V ukazni vrstici katerega koli vozlišča izvedemo ukaz `crm_mon -1r -A`. S tem pridobimo vse potrebne informacije o delovanju gruče, kar je prikazano na sliki 5.5.

```
Online: [ zabbix1 zabbix2 ]

Full list of resources:

 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):       Started zabbix1
     vip-rep    (ocf::heartbeat:IPaddr2):       Started zabbix1
 Master/Slave Set: msPostgresql [pgsql]
     Master: [ zabbix1 ]
     Slaves: [ zabbix2 ]
 Resource Group: ha-zabbix
     ha-ip        (ocf::heartbeat:IPaddr):        Started zabbix1
     ha-zabbix_server     (lsb:zabbix-server):    Started zabbix1

Node Attributes:
* Node zabbix1
    + master-pgsql                     : 1000
    + pgsql-data-status                : LATEST
    + pgsql-master-baseline            : 0000000037000090
    + pgsql-status                     : PRI
* Node zabbix2
    + pgsql-data-status                : STREAMING|ASYNC
    + pgsql-status                     : HS:async
```

Slika 5.5: Stanje sistema v visoki razpoložljivosti

Vidimo, da vsi potrebni viri tečejo na vozlišču zabbix1, vendar je vozlišče zabbix2 v stanju pripravljenosti. Za test preklopa z ukazom `shutdown -h now` izklopimo vozlišče zabbix1 in ponovno preverimo status gruče.

```
Online: [ zabbix2 ]
OFFLINE: [ zabbix1 ]

Full list of resources:

 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):       Started zabbix2
     vip-rep    (ocf::heartbeat:IPaddr2):       Started zabbix2
 Master/Slave Set: msPostgresql [pgsql]
     Master: [ zabbix2 ]
     Stopped: [ zabbix1 ]
 Resource Group: ha-zabbix
     ha-ip        (ocf::heartbeat:IPaddr):        Started zabbix2
     ha-zabbix_server     (lsb:zabbix-server):    Started zabbix2

Node Attributes:
* Node zabbix2
    + master-pgsql                     : 1000
    + pgsql-data-status                : LATEST
    + pgsql-master-baseline            : 000000003800020
    + pgsql-status                     : PRI
```

Slika 5.6: Stanje sistema po izklopu vozlišča zabbix1.

Na sliki 5.6 vidimo, da je uspel preklop vseh virov in da je Zabbix sistem ponovno na voljo na vozlišču zabbix2. Ker za dostop uporabljamo privzeta imena, ki so povezana z navideznimi naslovi, tako ne potrebujemo nobenega dodatnega nastavljanja in z uporabniškega vidika je delovanje popolnoma nemoteno, s čimer tudi dosežemo zastavljene cilje.

Poglavje 6

Zaključek

V nalogi smo uspešno opisali, naložili in nastavili vse pakete, ki skupaj sestavljajo visokorazpoložljiv Zabbix sistem. S testnimi primeri smo dokazali, da za visokorazpoložljiv sistem, ki je namenjen nadzoru omrežnih naprav in storitev, ne potrebujemo dragih produktov, ki jih ponujajo velike programerske hiše, saj lahko s pomočjo odprtokodnih produktov rešitev implementiramo sami.

Opisana rešitev je zgrajena modularno iz odprtokodnih paketov in standardiziranih protokolov opisanih v teoretičnem delu diplome. Če bi želeli sistem prilagoditi oziroma vgraditi obstoječ sistem, ki ni popolnoma združljiv z našo rešitvijo, lahko določeno funkcionalnost preprosto zamenjamo. Ker se ne zanašamo na postopke prirejene izključno naši rešitvi, lahko enostavno zamenjamo podatkovno bazo s katero od podprtih alternativ in le prilagodimo nastavitve preklopa. Ravno tako lahko nadomestimo Zabbix nadzorni sistem s katero od konkurenčnih rešitev.

Implementiran sistem ima še veliko možnosti za izboljšave in nadaljni razvoj. V nadaljevanju bi se lahko posvetili odpornosti preklopa, nastavitvi obveščevalnega sistema, porazdeljevanju bralnih poizvedb med aktivnim in pasivnim strežnikom ali v gručo dodali dodatna vozlišča. Omenjene izboljšave so izven obsega diplomske naloge, vendar predstavljajo izziv za prihodnost. Nenazadnje bi veljalo pregledati še kakšne druge sisteme za nadzor

in z njimi preveriti možnosti visokorazpoložljivega delovanja.

Literatura

[1] Andrew Beekhof. Pacemaker architecture. `http://clusterlabs.org/doc/en-US/Pacemaker/1.1/html/Clusters_from_Scratch/_pacemaker_architecture.html`, 2012. Pogledano 21.6.2014.

[2] S. Budrean, Yanhong Li, and B.C. Desai. High availability solutions for transactional database systems. In *Database Engineering and Applications Symposium, 2003. Proceedings. Seventh International*, pages 347–355, July 2003.

[3] J. Case, Mark Fedor, Martin L. Schoffstall, and James Davin. Simple Network Management Protocol (SNMP). RFC 1157, Internet Engineering Task Force, May 1990. http://tools.ietf.org/html/rfc1157.

[4] The Corosync Cluster Engine Community. The corosync cluster engine. `http://www.corosync.org`. Pogledano 19.7.2014.

[5] Raima Inc. High-availability database (ha db). `http://raima.com/rdme-high-availability-database`. Pogledano 12.7.2014.

[6] Issariyapat, C. and Pongpaibool, P. and Mongkolluksame, S. and Meesublak, K. Using nagios as a groundwork for developing a better network monitoring system. In *Technology Management for Emerging Technologies (PICMET), 2012 Proceedings of PICMET '12:*, pages 2771–2777, July 2012.

[7] J. Case and K. McCloghrie and M. Rose and S. Waldbusser. Introduction to Community-based SNMPv2. RFC 1901, Internet Engineering Task Force, January 1996. http://tools.ietf.org/html/rfc1901.

[8] J. Case and K. McCloghrie and M. Rose and S. Waldbusser. Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1902, Internet Engineering Task Force, January 1996. http://tools.ietf.org/html/rfc1902.

[9] D. Kundu and S.M.I. Lavlu. *Cacti 0.8 Network Monitoring*. From Technologies to Solutions. Packt Publishing, 2009.

[10] Bob Mortensen. Transparent failover. *Computer Technology Review*, 17(6):42–44, 06 1997.

[11] Nakamura, N. and Kashimura, N. and Motomura, K. CMIP to SNMP translation technique based on rule description. In *Computer Communications and Networks, 1995. Proceedings., Fourth International Conference on*, pages 266–271, Sep 1995.

[12] Zabbix SIA. Zabbix concepts. `https://www.zabbix.com/documentation/2.2/manual/concept`, October 2012.

[13] Stamou, K. and Kantere, V. and Morin, J.-H. SLA data management criteria. In *Big Data, 2013 IEEE International Conference on*, pages 34–42, Oct 2013.

[14] Steven C. Dake, Christine Caulfield, and Andrew Beekhof. The corosync cluster engine. In *Linux Symposium*, volume 85, 2008.

[15] Mani Subramanian. *NETWORK MANAGEMENT: Principles and Practice*. Addision-Wesley, 2000.

[16] Diane Teare. Structuring and Modularizing the Network with Cisco Enterprise Architecture - Network Management Protocols and

Features. `http://www.ciscopress.com/articles/article.asp?p=1073230&seqNum=4`, jun 2008. Pogledano 12.8.2014.

[17] Uri Blumenthal, Bert Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). RFC 2574, Internet Engineering Task Force, April 1999. http://tools.ietf.org/html/rfc2574.

[18] S. Waldbusser. Remote Network Monitoring Management Information Base. RFC 1757, Internet Engineering Task Force, Februar 1995. http://tools.ietf.org/html/rfc1757.

[19] S. Waldbusser, R. Cole, C. Kalbfleisch, and D. Romascanu. Introduction to the Remote Monitoring (RMON) Family of MIB Modules. RFC 3577, Internet Engineering Task Force, Avgust 2005. http://tools.ietf.org/html/rfc3577.

[20] Larry Walsh. *SNMP MIB Handbook: Essential Guide to Mib Development, Use, and Diagnosis*. Wyndham Press, 2008.

[21] Harald Zottmann. The corosync cluster engine. `http://www.cellsoft.de/telecom/cmip.htm`, Apr 2004. Pogledano 20.8.2014.